Random number generator using compression

Patent Number: EP1223506
Publication date: 2002-07-17
Inventor(s): SMEETS BEN (SE); ANDERSSON STEFAN (SE)
Applicant(s): ERICSSON TELEFON AB L M (SE)
Requested Patent: ☐ EP1223506
Application Number: EP20010610006 20010116
Priority Number(s): EP20010610006 20010116
IPC Classification: G06F7/58
EC Classification:
Equivalents:
Cited patent(s): US5757923; US5606322; EP0136681; US6076097; US5778069

Abstract

This invention relates to a random number generator that receives a number of input bits based on at least one random or pseudo-random source. The bits are compressed according to a compression scheme like Ziv-Lempel in order to enhance the entropy and randomness of the random source(s). Additionally, any eventual problem regarding drifting statistical characteristics, e.g. due to temperature change, aging of components, etc. is minimised/eliminated. In this way a random number generator is provided that generates high quality random numbers securely and efficiently. This invention also relates to a corresponding method for generating at least one random number based on a hardware source/unit.

Description

[0001] The present invention relates to a random number generator adapted to receive a number of input bits based on at least one random or pseudo-random source, said generator comprising

memory means comprising a state of bits,
calculating means for applying a given function to at least one bit of said state resulting in at least one random number.

[0002] The present invention also relates to a method for generating at least a random number comprising the steps of

receiving a number of input bits based on at least one random or pseudo-random source,
storing in a memory a state of bits,
applying a given function to at least one bit of said state resulting in at least one random number.

[0003] Additionally, the invention relates to a computer system for performing the method according to the invention and a computer readable-medium comprising a program, which may cause a computer to perform the method of the present invention.

[0004] Random numbers or pseudo random numbers (used interchangeably in the following) are used in many cryptographic and communication applications to provide randomly appearing symbols that e.g. may be used as session keys, random challenges, initialisation vectors, etc.

[0005] Unfortunately, random number generation is a very hard task to accomplish, for example general-purpose processors are not good at generating random numbers due to their intrinsic deterministic behaviour. One solution is to use the physical characteristics of a hardware entity/unit as a source of random input.

[0006] Additionally, state of the art cryptographically secure random number generators with a high level of unpredictability are computationally very demanding and are based on known computationally hard problems even if they are based on some characteristics of a hardware module. An account of various methods to produce random numbers is e.g. given in Chapter 5 of "The Handbook of Applied Cryptography, by A.J. Menezes, P.C. van Oorschot, and S.A. Vanstone, CRC Press 1997", which is incorporated herein by reference in its entirety.

[0007] Cryptographically secure random number generator based on computationally hard problems require very complex operations that are expensive in terms of hardware, power consumption and/or computational/execution time which is a problem especially in portable devices like a mobile phone, a PDA etc. Examples of such generator may be found in Section 5.5 of "The Handbook of Applied Cryptography, by A.J. Menezes, P.C. van Oorschot, and S.A. Vanstone, CRC Press 1997".

[0008] Examples of sources for random or pseudo-random data are: a current internal state of a hardware module/unit/device, date, time, free available memory, elapsed time between emission of particles during radioactive decay, thermal noise from semiconductor diodes or resistors, etc. or in general any source with a high degree of unpredictability/entropy.

[0009] Hardware-based random number generators in general also have some disadvantages like it is hard to find a hardware source with enough entropy and/or randomness and it is hard to generate random bits based on a hardware source at a sufficiently high rate since the hardware generally does not change state, parameters, etc. fast enough for being used as suitable input in random and/or pseudo random generation for some applications. Another problem is that the physical hardware source's statistical characteristic(s) changes over time as a result of environmental influences (e.g. temperature), aging and/or wear out, so that an initially 'good' random source may become less random due to these effects.

[0010] The ANSI X.9.17 (see "The Handbook of Applied Cryptography, by A.J. Menezes, P.C. van Oorschot, and S.A. Vanstone, CRC Press 1997", page 173) uses a cryptographic primitive, i.e. DES, to

increase the unpredictability of the output of a random number generator. The DES encryption algorithm is a rather expensive (in terms of hardware complexity and/or number of CPU operations) solution and additionally still suffers from the undesired side effect that the source of random inputs may have drifting statistical characteristics.

[0011] The US patent specification no. 5,778,069 discloses a pseudo random number generator that assembles multiple bits from a number of different sources into an input bit string. A hash value is computed by using a hashing function on the input bit string, and a stream generator uses this hash value to produce an output bit string of random bits. The different sources may e.g. be an internal source, like a state register in the generator itself, an external source like a computer supplying an operating parameter (like current date, time, free available memory, etc.), and another external source like a program/application supplying bits relating to the execution of the program/application.

[0012] An object of the invention is to provide a random number generator that generates high quality random numbers securely and efficiently.

[0013] This object is achieved by a random number generator of the aforementioned kind that further comprises

compression means for compressing said input bits resulting in a compressed bit string, which is stored in said memory means as at least a part of said state.

[0014] Hereby, a random number generator is provided, which greatly enhances the entropy of the random source(s) by using compression on bits representing the source(s). Additionally, any eventual problem regarding drifting statistical characteristics, e.g. due to temperature change, aging of components, etc. is minimised/eliminated.

[0015] The compression/coder means removes redundant data from the random source(s) and maximises the entropy of the input bits thereby enhancing the random quality of the generated random or pseudo-random symbol/sequence.

[0016] Preferably, the compression means are adaptive, i.e. capable of adapting to the varying statistical characteristics of the input data. Such means are generally known as universal source coding means.

[0017] The input bits are e.g. obtained by sampling the random source(s) at a given rate.

[0018] The random source(s) may be a combination of random or pseudo-random sources.

[0019] According to a preferred embodiment, the at least one random or pseudo-random source comprises at least one hardware source and said input bits represents at least one physical feature of said at least one hardware source.

[0020] Hereby, at least one hardware-based random or pseudo-random source is provided, which normally provides the generator with high quality random and/or pseudo-random input. The compression means ensures that the input stays high quality with respect to randomness and entropy thereby avoiding statistical drifting.

[0021] In accordance with one embodiment of the generator, the state further comprises at least a part of a state of a Finite State Machine (FSM).

[0022] In this way, the memory means may change state and thereby input for the calculating means even if the random or pseudo-random source(s) do not. This enables the use of random and/or pseudo-random sources, which are relatively slower, as input, while still having a relatively fast changing state, since the overall state, i.e. the state comprising a FSM and random/pseudo-random input, will change every clock cycle.

[0023] Additionally, it is generally simpler to ensure that a FSM does not have local cycles of possible states.

[0024] In accordance with another embodiment, the Finite State Machine (FSM) is a linear feedback shift register (LFSR) or a non-linear feedback shift register (FSR).

[0025] The LFSR is a FSM with good statistical properties and thereby increases the overall randomness and entropy in the state of the memory means.

[0026] A non-linear FSR have even better statistical properties, which increases the security at the cost of a small increase in complexity.

[0027] In a preferred embodiment, the memory means comprises a cyclic buffer having a length of at least one bit.

[0028] Hereby, very simple and efficient part of the memory means is provided.

[0029] In an preferred embodiment, the calculating means are adapted to compute at least one hash value resulting in at least one random number using a hashing function.

[0030] A hash function makes it virtually impossible to determine the input of the function on the basis of the output, which greatly increases the security of the generated random numbers/sequence.

[0031] Alternatively, the calculating means comprises feedback to the FSM, thereby increasing the output rate of random numbers.

[0032] In yet another embodiment, the hashing function is any one of:

SHA-1,
RIPEMD-160,
MD5 or
any other suitable hashing or similar function.

[0033] In a preferred embodiment, the compression means compresses input bits according to a Lempel-Ziv compression scheme.

[0034] The Lempel-Ziv algorithm is a well-known universal source coding method, i.e. it is adaptable, and is e.g. described in Ziv, Lempel, "A universal algorithm for sequential data compression", IEEE Transaction on Information Theory Vol. 23, no 5, p. 337-343, May 1977 and Ziv, Lempel "Compression of individual sequences via variable rate coding", IEEE Transactions on Information Theory, Vol. 24, no 5, p. 530-536, September 1978.

[0035] Alternatively, the compression means compresses the input bits according to another compression/encoding scheme/algorithm like PPM (partial pattern matching), Huffman, Tunstall etc. or other compression schemes used e.g. to obtain better compression, lower implementation costs, and/or execution speed.

[0036] In one embodiment, the random number generator is used in a portable device.

[0037] In a preferred embodiment, the portable device is a mobile telephone.

[0038] Another object of the invention is to provide a method that generates high quality random numbers secure and efficiently.

[0039] This object is achieved by a method of the aforementioned kind, which further comprises the step of

compressing said input bits resulting in a compressed bit string, which is stored in said memory as at least a part of said state.

[0040] According to a preferred embodiment, the at least one random or pseudo-random source comprises at least one hardware source and said input bits represents at least one physical feature of said at least

one hardware source.

[0041] According to one embodiment of the method, the state further comprises at least a part of a state of a Finite State Machine (FSM).

[0042] In accordance with another embodiment, Finite State Machine (FSM) is a linear feedback shift register (LFSR) or a non-linear feedback shift register (FSR).

[0043] In a preferred embodiment, the memory means comprises a cyclic buffer having a length of at least one bit.

[0044] In a preferred embodiment, the calculating means are adapted to compute at least one hash value resulting in said random number using a hashing function.

[0045] In yet another embodiment, the hashing function is any one of:

SHA-1,
RIPEMD-160,
MD5 or
any other suitable hashing or similar function.

[0046] In a preferred embodiment, the compression means compresses the input bits according to a Lempel-Ziv compression scheme.

[0047] In one embodiment, the method is used in a portable device.

[0048] In a preferred embodiment, the method is used in a mobile telephone.

[0049] The method and embodiments thereof correspond to the random number generator and embodiments thereof and has the same advantages for the same reasons why they are not described again.

[0050] Further, the invention relates to a computer-readable medium having stored thereon instructions for causing a processing unit or a computer system to execute the method described above and in the following. A computer-readable medium may e.g. be a CD-ROM, a CD-R, a DVD RAM/ROM, a floppy disk, a hard disk, a smart card, a network accessible via a network connection, a ROM, RAM, and/or Flash memory, etc. or generally any other kind of media that provides a computer system with information regarding how instructions/commands should be executed.

[0051] Hereby, when a computer is caused to retrieve electronic information - as a consequence of the contents of a computer-readable medium as described above - the advantages mentioned in connection with the corresponding method according to the invention are achieved.

[0052] Finally, the invention relates to a computer system comprising means adapted to execute a program, where the program, when executed, causes the computer system to perform the method according to the invention thereby obtaining the above mentioned advantages and/or effects.

[0053] By computer system is meant e.g. a system comprising one or more processor means, like a specialised or general purpose CPU or the like, which may be programmed/instructed at one time or another in such a way that the computer executes the method according to the invention fully or in part.

[0054] The present invention will now be described more fully with reference to the drawings, in which

Figure 1 illustrates schematically a functional block diagram of a random number generator according to the invention;
Figure 2 illustrates schematically a functional block diagram of a random number generator according to one embodiment of the invention;
Figure 3 illustrates a generalised functional block diagram of a random number generator according to another embodiment of the invention;

Figure 4 shows a flowchart illustrating the method according to the present invention;
Figure 5 shows a preferred embodiment of the invention, which may contain the random number generator and/or use the method according to the present invention;
Figure 6 illustrates an example of how the input bits are affected by the compression means;
Figure 7 illustrates a more detailed example of how the input bits may be affected by the compression means and what output is generated.

[0055] Figure 1 illustrates schematically a functional block diagram of a random number generator according to the invention. Shown are a number of input bits (101) representing information of one or more random and/or pseudo-random sources (not shown). The input bits (101) may e.g. be obtained by sampling the source(s) at an appropriate rate. In a preferred embodiment at least one of the sources is a hardware-based source.

[0056] Compression means (102) receive the input bits (101) and compresses them according to a given compression scheme or method like the ones described above. An example of compression of the input bits (101) is given in connection with Figures 6 and 7. The compression means (102) removes redundant information from the input bits (101) thereby enhancing the entropy and the random properties of the input bits (101). The compression means (102) also removes the effect of drifting characteristics when using a hardware-based source.

[0057] Preferably, the compression means uses a Lempel-Ziv compression scheme.

[0058] Alternatively, the compression means compresses the input bits according to another compression/coding scheme/algorithm like PPM (partial pattern matching), Huffman, Tunstall etc. or other compression schemes used e.g. to obtain better compression, lower implementation costs, and/or execution speed.

[0059] The result from the compression means (102) is stored as at least a part of a state in memory means (103).

[0060] In one embodiment, the memory means (103) is a cyclic buffer with a length of at least one bit. In a cyclic buffer the last element of the buffer becomes the first element of the buffer when the buffer is full thereby avoiding overloading/overwriting.

[0061] Alternatively, the state of the memory means (103) further comprises a Finite State Machine (FSM) or a copy of the state of a FSM, which ensures that the state of the memory means (103) is changed at every clock signal even though the random and/or pseudo random sources (and thereby the output from the compression means (102)) does not change. The FSM may be any suitable linear feedback shift register (LFSR) or a non-linear feedback shift register (FSR), and one example of a FSM is described in more detail later in connection with Figure 2 and 3. The FSM is shifted at least one step.

[0062] Calculation means (104) calculates a given function of the state of the memory means (103) in order to derive at least one random or pseudo random number (105). Preferably, a hashing function is used by the calculation means (104) in order to derive the random number(s) (105), which enhances the security of the process, since a hashing function ensures that it is almost impossible or at least very hard to derive the input of the hashing function on the basis of the output.

[0063] Alternatively, other functions may be used by the calculation means (104) in order to ensure the output of a random or pseudo-random number or sequence.

[0064] Alternatively, the calculation means (104) may provide feedback to the memory means (103) or more specifically to the FSM of the memory means (103), which increases the output rate of the random or pseudo-random number(s)/sequence, e.g. may the output from the calculation/hashing means (104) be split in two not necessarily equal parts where one part is the random number/sequence and the other part is used to control the FSM/LFSR.

[0065] Figure 2 illustrates schematically a functional block diagram of a random number generator according to one embodiment of the invention. Shown are a number of random or pseudo-random bits (201) obtained from a hardware source (not shown) e.g. by sampling one or more physical parameters

from the source at a given rate. The bits (201) preferably represent one or more physical characteristics of the hardware source/unit.

[0066] Alternatively, the bits are obtained on the basis of other not necessarily hardware-based random or pseudo-random sources.

[0067] The bits (201) are compressed by compression means (202) in order to remove redundant information, maximise the entropy and remove any effects due to drifting of statistical characteristics of the hardware-based source/unit. This ensures that the bits (201) after compression are as random as possible.

[0068] Preferably, the compression means uses a Lempel-Ziv compression scheme.

[0069] Alternatively, the compression means compresses the input bits according to another compression scheme/algorithm like PPM (partial pattern matching), Huffman, Tunstall etc. or other compression schemes used e.g. to obtain better compression, lower implementation costs, and/or execution speed.

[0070] The compressed bits are stored in a memory preferably a cyclic buffer (203). Preferably the memory further comprises at least part of a state of a FSM (204). The FSM (204) ensures that the memory state changes at every clock cycle or for every random number generation even though the hardware source does not, which normally will happen quite often since the hardware unit's physical features changes relatively slower than a FSM circuit.

[0071] Additional randomness of high quality is introduced in the memory means, if the state of the FSM (204) has good statistical/random properties.

[0072] The FSM will as all finite state machines eventually cycle, i.e. return to a previously produced state.

[0073] In most applications the FSM will, as shown in this example, operate over a binary symbol field, i.e. the symbols will be "0" and "1". However, other symbol fields could be used just as well as is already well known from the theory of FSM, linear feedback shift registers, etc.

[0074] A linear feedback shift register (LFSR) is one example of a FSM with good statistical properties, which thereby increases the overall randomness and entropy in the state of the memory means.

[0075] The LFSR is advantageously chosen in such a way that the generated sequence of states will cycle through all the non-zero states, which can be realised by choosing the feedback connections corresponding to a primitive polynomial over the symbol field, see e.g. "The Handbook of Applied Crypotography, by A.J. Menezes, P.C. van Oorschot, and S.A. Vanstone, CRC Press 1997, page 195-196".

[0076] The security can even be increased at the cost of a small increase in complexity when the feedback of the LFSR is modified to generate a full-length sequence like a de Bruin sequence, i.e. the LFSR will cycle through all the possible states including the zero state (comprising all "0").

[0077] A non-linear feedback shift register (FSR) is a FSM with even better statistical properties, which increases the security even more at the cost of a small increase in complexity of the implementing hardware.

[0078] The complete state of the memory means (203; 204) is then used as input to a hashing means (205) in order to enhance the security even further resulting in one or more random or pseudo-random bits.

[0079] Applying a hashing or similar "one-way" function to the bits of the memory means (203; 204) ensures that it is almost impossible to determine the relationship between the input and the output of the hashing means (205), which increases the security of the random or pseudo-random number/sequence.

[0080] Alternatively, a part of the state of the memory means (203; 204) is fed to the hashing means (205).

[0081] Examples of hashing functions are SHA-1, RIPEMD-160, MD5 or in general any other suitable hashing or similar one-way function.

[0082] Figure 3 illustrates a generalised functional block diagram of a random number generator according to another embodiment of the invention. This figure corresponds to Figure 2 apart from that a plurality of N random and/or pseudo-random sources (201; 201') are shown instead of just a single one as in Figure 2.

[0083] The random sources (201; 201') may comprise any hardware-based or non hardware-based source capable of being sampled/sensed in one way or another in order to generate symbols representing at least one random and/or pseudo-random number. By random number is meant a number representing a 'truly' random characteristic often sensed/sampled physical parameter(s), and by pseudo-random number is meant a number representing parameters based on a deterministic (although complex) unit like an internal state of a hardware circuit/unit, a clock circuit, CPU, memory state, etc.

[0084] Some examples of random sources are: elapsed time between emission of particles during radioactive decay, thermal noise from semiconductor diodes or resistors, etc. or in general any source with a high degree of unpredictability/entropy based on a physical characteristics.

[0085] The radioactive source and sensor may e.g. be encapsulated in hard plastic or the like so no third party can determine the parameter(s) thereby breaching the security.

[0086] Some examples of pseudo-random sources are: a current internal state of a hardware module/unit/device, date and/or time in a computer system, free available memory, memory state (e.g. in a FSM), speed change of harddisk while performing a reading and/or writing operation, a combination of a number of unsynchronised clocks, etc.

[0087] In this way, even better random and/or pseudo-random input is obtained by the generator thereby enhancing the quality of the output random/pseudo-random number(s).

[0088] Figure 4 shows a flowchart illustrating the method according to the present invention. The method is initialised and started at step (400). At step (401) a number of bits representing parameters of one or more random and/or pseudo-random sources are obtained e.g. by sampling the source(s). Preferably, at least one source is a hardware-based source, but other sources may be used just as well as described earlier.

[0089] At step (402) a FSM is shifted at least one step to another state. The FSM may be any suitable linear feedback shift register (LFSR) or a non-linear feedback shift register (FSR) as described above. This ensures that the input basis (and thereby the final output) of the random and/or pseudo-random number generation is changed at every clock-cycle independently of a change in the source(s).

[0090] At step (403) the bits obtained in step (401) are compressed and put in a memory element preferably a cyclic buffer.

[0091] The compression is preferably done according to a Lempel-Ziv compression scheme, but other compression schemes/algorithms like PPM (partial pattern matching), Huffman, etc., but other compression schemes used e.g. to obtain better compression, lower implementation costs, and/or execution speed may also be used.

[0092] At step (404) the compressed bits and the bits of the present FSM state are combined into a single bit string or the like.

[0093] Alternatively, instead of combining the compressed bits and the FSM state they could be stored as a part of a single memory element.

[0094] At step (405) the combined compressed bits and the bits of the FSM state are used as input in a hashing function in order to derive at least one random and/or pseudo-random number. The hashing function ensures that it is almost impossible or at least very hard to derive the input of the hashing function on the basis of the output, which enhances the security of the generated random and/or pseudo-random numbers.

[0095] The hashing function may be any one of SHA-1, RIPEMD-160, MD5 or in general any other suitable hashing or similar function.

[0096] At step (406) a test is made whether further random number(s) are to be generated. If this is the case the method loops back and performs step (401) to (406) once again.

[0097] Alternatively, step (402) may also be executed e.g. after step (404) instead of before step (403) or in any other place in the sequence of steps as long as the FSM is shifted at least one step for every derivation of the final random and/or pseudo-random number(s).

[0098] Alternatively, steps (402) and (403) may also be executed in parallel or in reverse order (403 before 402).

[0099] Figure 5 shows a preferred embodiment of the invention, which may contain the random number generator and/or use the method according to the present invention. Shown is a mobile telephone (501) having display means (504), a keypad (505), an antenna (502), a microphone (506), and a speaker (503). By including the random number generator and/or use the method, high quality random number generation is provided with a relatively small complexity in terms of hardware and thereby small power consumption and/or computational/execution time.

[0100] Figure 6 illustrates an example of how the input bits are affected by the compression means. Shown are bits (601) obtained from at least one random and/or pseudo-random source e.g. hardware-based source(s). In this example the distribution of "0" is approximately 25% and the distribution of "1" is approximately 75% which means that the source(s) have generated three times as many "1" as "0" e.g. due to drifting statistical characteristics. This is not an optimal distribution of random bits since one symbol occurs more often than the other, which reduces the security and randomness of the generated random number sequence.

[0101] The bits (601) are compressed by compression means (602) according to a Huffman compression scheme or according to another suitable scheme e.g. the ones described earlier, which gives the output bits stored in memory means (603) e.g. as a state where the distribution of "0" is approximately 50% and the distribution of "1" is approximately 50%, since the compression means remove any redundant information from the bits (601) thereby maximising the entropy.

[0102] Alternatively, the bits (601) further comprises a state of a FSM as described above, i.e. the bits (601) is made up of bits from a random and/or pseudo-random source(s) and of bits of the FSM.

[0103] Figure 7 illustrates a more detailed example of how the input bits may be affected by the compression means and what output is generated.

[0104] Shown are compression means (702), receiving an input sequence u ELEMENT {0,1}, and memory means comprising a cyclic buffer (703), comprising a state of code words, and a FSM like a maximum length LFSR (704) also comprising a state. The memory means also comprises MOD 2 adders (705) for combining at least part of the state of the buffer (703) and at least part of the state of the LFSR (704). In this example the LFSR (704) is of length 4 and the buffer (703) is of length 12 but other types and lengths of buffers and FSMs may be used. The buffer (703) and LFSR (704) are shown in their initial state.

[0105] The states of the buffer (703) and the LFSR (704) are in this embodiment used as input (706) in the calculation means/hashing means (not shown).

[0106] The input sequence u has statistical properties, which may be described by a Markov chain and have a corresponding entropy value.

[0107] In this particular example, the compression/coder means (702) compresses an input sequence u by a fixed source code, like a Tunstall code e.g. having the message set M ELEMENT {"1"; "10"; "100", "000"}, i.e. the sequence u of raw/random data is parsed into blocks corresponding to the message sets of the code (C0 C1). In this example the Tunstall code encodes messages as (time of M is increasing from right to left):
Columns=3

Head Col 1: M
Head Col 2: C0
Head Col 3: C1

111 ·
101 0
10001
00000

[0108] That is the input sequence/string u are parsed until a "1" or three consecutively "0" are met. If a "1" is met right away "11" are outputted as a codeword; if a single "0" is met before (in time) the "1", then "10" are outputted; if two "0" are met before (in time) the "1", then "01" is outputted; and finally if three "0" are met consecutively without meeting a "1", then "00" is outputted.

[0109] For example will the following input sequence û (with time increasing from right to left), where the sequence has been separated by a space in order to better show the message sets in the input string:
û: 000 000 10 000 1 000 100 1 100 10 1 100 1 generate the following encoded string as output(with time increasing from right to left):
00 00 01 00 11 00 10 11 10 01 11 10 11

[0110] Note however, that the source encoder/compression means in this particular example not always will emit a codeword for every input symbol.

[0111] The variable length input to fixed length output may be preferred, since the buffer clocking is easier when only fixed size steps are needed.

[0112] The generated sequence of code words are put in the cyclic buffer (703) as described later.

[0113] Alternatively, other source encoder/compression means may be used e.g. the ones mentioned earlier or in general any other suitable means, which ensures a high degree of entropy and/or low redundancy after processing the input sequence.

[0114] The LFSR (704) preferably has the following state transition behaviour:

[0115] A state comprising "0000", which transitions to it self, and another state, which transitions according to the following:
Columns=2
0"1000"
1" 1100"
2"1110"
3"1111"
4"0111"
5"1011"
6"0101"
7"1010"
8"1101"
9"0110"
10"0011"
11"1001"
12"0100"
13"0010"
14"0001"
where the state "0001" (14) transitions to state "1000" (0) and the state cycle (0 - 14) is repeated.

[0116] Together these two cycles of length 1 and 15, respectively exhaust the complete state space of the FSM/LFSR (704).

[0117] In operation the compression means (702) may operate like the following:

1) first shift buffer (703) and LFSR (704)
2) read symbol from the sequence u
3) if no complete message block (M) then goto 1 else
if number of shifts since last codeword < 2 then shift buffer (703) once more

write codeword (C0 C1) in the first two positions of the buffer.

[0118] Alternatively, the buffer may be clocked exactly twice between two source code words being written.

[0119] In the following the states of the buffer (703) and the LFSR (704) are shown while receiving the example input string û. The initial values/states of the buffer (703) and LRSR (704) are shown in parenthesis.

[0120] The parity of the combination of the two states is also shown.
Columns=6

Head Col 1: ûi
Head Col 2:
Head Col 3: Buffer (000000000000)
Head Col 4:
Head Col 5: LFSR (1000)
Head Col 6: Parity
1
*
11000000000011000
001100000000011101
000110000000011110
101011000000001110
1
*
11010110000010110
001101011000001011
110010101100010101
001001010110011010
000100101011001101
101010010101100110
1
*
110101001010
**
00011
001101010010110001
0101101010010
**
01001
101011010100100101
0101011010100
**
10010
001010110101001001
000101011010100101
1
*
110010101101
**
10011
0111001010110
**
11001
001110010101111100
0001110010101
**
01111

0100111001010
**
00110
110001110010110010
0110001110010
**
11000
001100011100111101
0001100011100
**
01110
000011000111010110
000001100011101011
0000001100011
**
00101
where * denotes an extra shift and

** denotes that the LFSR state is influenced by the buffer.

[0121] The states of the buffer and LFSR is used as input to a hashing or similar one-way function in order to greatly increase the security of the generated random numbers/sequence, since it becomes virtually impossible to determine the input of the function on the basis of the output.

[0122] Alternatively, the hashing or similar function may use only a part of the states.

[0123] As a very simple example, the following hash function is considered:
h(x1, x2, ... x16) = SIGMA ' xi = (x1 + x2 ... + x16) MOD 2
where i = 1...16 and SIGMA ' denotes the sum MOD 2.

[0124] In this very simple example the h is equal to the parity of the combined states as indicated in the above table.

[0125] In real applications h would be a one-way hash function like MD-5, SHA-1, RIPEMD-160 or any other suitable hashing or similar function.

[0126] It may be preferred that the length of the buffer is a relative prime to the block size of the code word (not the case in the above example for simplicity).

[0127] In a preferred embodiment, the new code words are not written into the buffer thereby overwriting old data but to "add" the code word data into at least one position/state in the buffer. Adding may e.g. be addition MOD 4 or XOR-ring of four bits.

Claims

1. A random number generator adapted to receive a number of input bits (101; 201, 201'; 601) based on at least one random or pseudo-random source, said generator comprising

memory means (103; 203, 204; 603; 703, 704) comprising a state of bits,
calculating means (104, 205) for applying a given function to at least one bit of said state resulting in at least one random number (105)
characterized in that said generator further comprises

compression means (102; 202; 602; 702) for compressing said input bits (101; 201, 201'; 601) resulting in a compressed bit string, which is stored in said memory means (103; 203, 204; 603; 703, 704) as at least a part of said state.

2. A random number generator according to claim 1, characterized in that said at least one random or pseudo-random source comprises at least one hardware source and said input bits represents at least one physical feature of said at least one hardware source.

3. A random number generator according to claims 1 - 2, characterized in that said state further comprises at least a part of a state of a Finite State Machine (FSM) (204).

4. A random number generator according to claim 3, characterized in that said Finite State Machine (FSM) (204) is a linear feedback shift register (LFSR) (704) or a non-linear feedback shift register (FSR).

5. A random number generator according to claims 1 - 4, characterized in that said memory means (103; 203, 204; 603; 703, 704) comprises a cyclic buffer (203; 703) having a length of at least one bit.

6. A random number generator according to claims 1 - 5, characterized in that said calculating means (104, 205) are adapted to compute at least one hash value resulting in said random number (105) using a hashing function.

7. A random number generator according to claim 6, characterized in that said hashing function is any one of:

SHA-1,
RIPEMD-160,
MD5 or
any other suitable hashing or similar function.

8. A random number generator according to claims 1 - 7, c haracterized in that said compression means compresses the input bits according to a Lempel-Ziv compression scheme.

9. A random number generator according to any one of the previous claims, characterized in that said random number generator is used in a portable device.

10. A random number according to claim 9, characterized in that said portable device is a mobile telephone (501).

11. A method of generating at least a random number comprising the steps of

receiving a number of input bits based on at least one random or pseudo-random source,
storing in a memory a state of bits,
applying a given function to at least one bit of said state resulting in at least one random number
characterized in that said method further comprises the step of

compressing said input bits resulting in a compressed bit string, which is stored in said memory as at least

a part of said state.

12. A method according to claim 11, characterized in that said at least one random or pseudo-random source comprises at least one hardware source and said input bits represents at least one physical feature of said at least one hardware source.

13. A method according to claims 11 - 12, characterized in that said state further comprises at least a part of a state of a Finite State Machine (FSM).

14. A method according to claim 13, characterized in that said Finite State Machine (FSM) is a linear feedback shift register (LFSR) or a non-linear feedback shift register (FSR).

15. A method according to claims 11 - 14, characterized in that said memory means comprises a cyclic buffer having a length of at least one bit.

16. A method according to claims 11 - 15, characterized in that said calculating means are adapted to compute at least one hash value resulting in said random number using a hashing function.

17. A method according to claim 16, characterized in that said hashing function is any one of:

SHA-1,
RIPEMD-160,
MD5 or
any other suitable hashing or similar function.

18. A method according to claims 11 - 17, characterized in that said compression means compresses the input bits according to a Lempel-Ziv compression scheme.

19. A method according to any one of the claims 11 - 18, characterized in that said method is used in a portable device.

20. A method according to any one of the claims 11 - 19, characterized in that said method is used in a mobile telephone.

21. A computer-readable medium having stored thereon instructions for causing a processing unit to execute the method according to any one of claims 11 - 20.

22. A computer system comprising means adapted to execute a program, where the program, when executed, causes the computer system to perform the method according to claims 11 - 20.

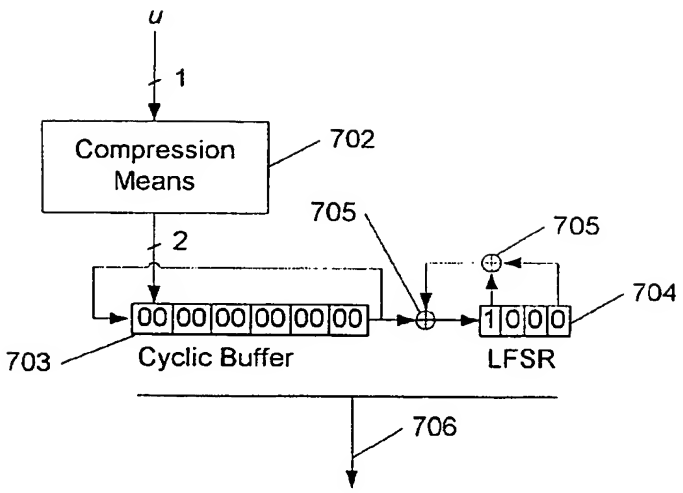

Figure 7

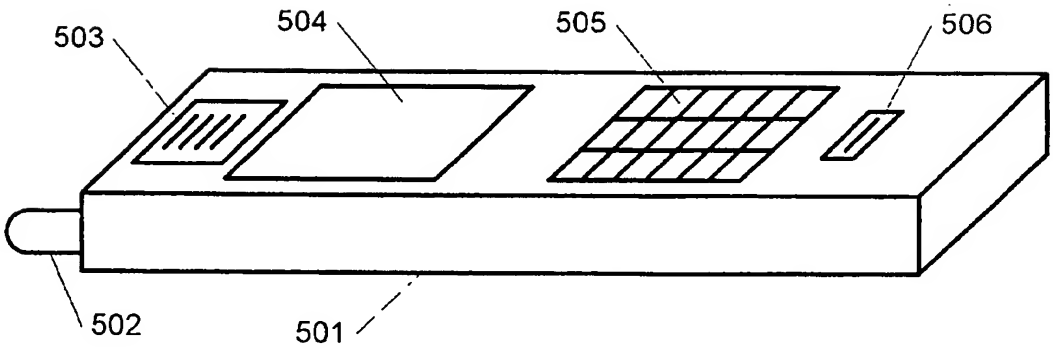

Figure 5

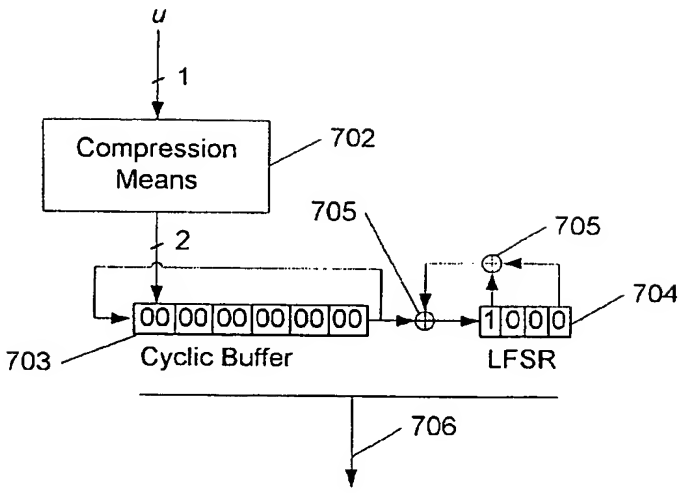

Figure 7

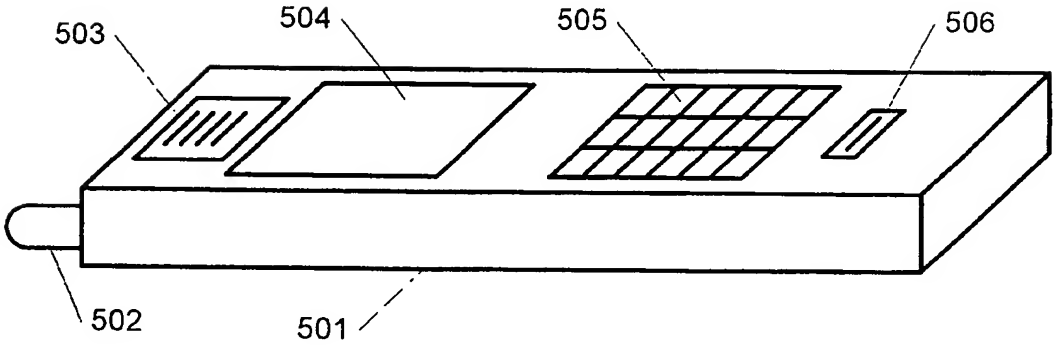

Figure 5